Quantitative Evaluation of Dynamic Platform Techniques as a Defensive Mechanism [⋆]

Hamed Okhravi, James Riordan, and Kevin Carter

MIT Lincoln Laboratory
{hamed.okhravi,james.riordan,kevin.carter}@ll.mit.edu

Abstract. Cyber defenses based on dynamic platform techniques have been proposed as a way to make systems more resilient to attacks. These defenses change the properties of the platforms in order to make attacks more complicated. Unfortunately, little work has been done on measuring the effectiveness of these defenses. In this work, we first measure the protection provided by a dynamic platform technique on a testbed. The counter-intuitive results obtained from the testbed guide us in identifying and quantifying the major effects contributing to the protection in such a system. Based on the abstract effects, we develop a generalized model of dynamic platform techniques which can be used to quantify their effectiveness. To verify and validate our results, we simulate the generalized model and show that the testbed measurements and the simulations match with small amount of error. Finally, we enumerate a number of lessons learned in our work which can be applied to quantitative evaluation of other defensive techniques.

Keywords: Dynamic platforms, platform diversity, quantitative evaluation, metrics, intrusion tolerance, moving target

1 Introduction

Developing secure systems is difficult and costly. The high cost of effectively mitigating all vulnerabilities and the far lesser cost of exploiting a single one creates an environment which advantages cyber attackers. New active cyber defense paradigms have been proposed to re-balance the landscape and create uncertainty for the attackers [1]. One such paradigm is active defenses based on dynamic platform techniques.

Dynamic platform techniques (or simply, dynamic platforms) dynamically change the properties of a computing platform in order to complicate attacks. Platform properties refer to hardware and operating system (OS) attributes such as instruction set architecture (ISA), stack direction, calling convention, kernel version, OS distribution, and machine instance. Various dynamic platform techniques have been proposed in the literature. Emulation-based techniques change the calling sequence and instruction set presented to an application [2]; multivariant execution techniques change properties

[⋆] This work is sponsored by the Department of Defense under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

such as stack direction or machine description using compiler generated diversity and virtualization [3–6]; migration-based techniques change the hardware and operating system of an application using containers and compiler-based checkpointing [7]; server diversification techniques rotate a server across multiple platforms and software stacks using network proxies [8]; self cleansing techniques change the machine instance by continuously rotating across many virtual machines and re-imaging the inactive ones [9–11].

Unfortunately, little work has been done on understanding and quantifying the impact of dynamic platforms on the security of a system. The impact of such techniques is often assumed to be intuitive and straight forward. Moreover, one cannot compare different features provided by different dynamic platforms in a quantitative way. For example, is it more effective to support multiple platforms that are running simultaneously and voting on the result (a.k.a. multi-instance), or to have one active platform, but support cleansing of the inactive ones (a.k.a. cleanup)?

In this work, we first identify the four major features proposed by different dynamic platforms in the literature. We then perform a set of experiments on a testbed with one such technique that is augmented to support these features in order to quantify its protection. The results from our testbed experiments are, in fact, counter-intuitive and complex. The complexity of the results suggest that various underlying effects contribute to such a system.

Based on our observations and the mathematical principles involved, we enumerate and analyze the various underlying effects in an abstract analysis of a dynamic platform system. To evaluate the completeness of our enumerated list of abstract effects, we develop a generalized model of dynamic platforms based on these effects and verify and validate the model by simulating the same experiments as the ones we performed on the testbed. The matching results and the small amounts of error validate our model and verify that we have at least correctly captured the main effects contributing to the protection provided by a dynamic platform. Finally, we enumerate a number of lessons learned that can be applied to the quantitative evaluation of other defensive techniques.

Our contributions are as follows:

- To the best of our knowledge, we perform the first quantitative evaluation of dynamic platforms as a defensive mechanism and illustrate the complexities and the counter-intuitive effects contributing to such a system. Moreover, we enumerate the major effects and their impacts.
- We develop a generalized model of dynamic platforms and simulate the results. We verify and validate the model by comparing the simulated results with the testbed experiments and show that they match closely.
- We demonstrate how testbed experiments, abstract analysis, and modeling and simulation can be used together to quantify the impact of defensive techniques. In our work, testbed experiments are used to uncover the complexities, abstract analysis is used to enumerate and describe such complexities, and modeling and simulation is used to check the completeness of the abstract analysis and to validate the results. We enumerate a number of lessons learned which can guide future evaluations of the defenses.

The rest of the paper is organized as follows. Section 2 provides a brief overview of dynamic platform techniques. Section 3 describes the threat model used throughout the paper. Section 4 discusses our testbed experiments and measurements performed on a real system. Section 5 discusses our abstract analysis approach and its results. Section 6 describes our generalized model of dynamic platforms. Section 7 presents the simulation results from the generalized model. Section 8 enumerates a number of lessons learned and discusses our findings. We discuss the related work in Section 9 before concluding the paper in Section 10.

2 Dynamic Platform Background

We briefly describe the defensive techniques based on dynamic platforms. We provide enough background for understanding the rest of the paper. More details about each technique can be found in its original publication.

Dynamic platform techniques change platform properties in order to make attacks more complicated [12]. They often rely on temporal changes (e.g. VM rotation), diversity (e.g. multivariant execution), or both (e.g. migration-based techniques) to protect a system. These techniques are often implemented using machine-level or operating system-level virtualization, compiler-based code diversification, emulation layers, checkpoint/restore techniques, or a combination thereof. Emulation-based techniques such as Genesis [2] often use an application-level virtual machines such as Strata [13] or Valgrind [14] to implement instruction set diversity. In some cases, multiple instances are executed and a monitor compares their results. Multivariant execution techniques such as Reverse stack [15] (also called N-variant systems [16]) use compiler-based techniques to create diverse application code by replacing sets of instructions with semantically equivalent ones. Migration-based techniques such as Talent [7] use operating system-level virtualization (containers) to move an application across diverse architectures and operating systems. A dynamic platform can also be achieved at a higher abstraction level by switching between different implementations of servers [8]. These techniques either do not preserve the state (e.g. a web server) or they preserve it using high level configuration files (e.g. DNS server). Finally, self-cleansing techniques such as SCIT [9] only change the current instance of the platform without diversifying it. The main goal, in this case, is bringing the platform to its pristine state and removing persistence of attacks.

We have identified four features that determine the protection provided by dynamic platform techniques. Later in our analysis, we show that these features can result in very different defensive benefits for each technique. The four features are:

Diversity A dynamic platform technique provides diversity if it changes the properties of the platform used for running the application. For example, the Reversed Stack [15] technique provides diversity because it changes the direction of stack growth whereas SCIT [9] does not because it rotates the service among homogeneous virtual machines.

Multi-Instance A technique is multi-instance if more that one platform instance is used to serve a transaction simultaneously. For example, multivariant execution [3]

Technique	Diversity	Multi-Instance	Limited Duration	Cleanup
SCIT [9]				✓
GA-Based Configuration [17]	✓		✓	
MAS [18]	✓			✓
Multivariant Execution [3]	✓	✓		
Reversed Stack [15]	✓	✓		
Talent [17]	✓		✓	
Machine desc. diversity [6]	✓		✓	
N-Variant System [16]	✓	✓		
Intrusion Tolerance for MCS [19]	✓	✓		
Intrusion Tolerant WS [8]	✓	✓		

Table 1: Features of some of the dynamic platform techniques

is a multi-instance technique because it runs a transaction on multiple different instances of the platform and compares the results, whereas Talent [7] is not, because it uses one instance at a time.

Limited Duration A technique has limited duration if the instance of the platform can change while processing a single transaction. Otherwise, we call it extended duration which means that the technique must finish processing a transaction before it can change the instance of the platform. For example, using genetic algorithms to change platform configurations [17] has limited duration because the the configuration can change while processing a transaction whereas moving attack surfaces [18] completes each transaction on the same instance on which it started (i.e. extended duration).

Cleanup A technique supports cleanup if each instance is wiped and imaged into a pristine state before it is used again. For example, SCIT [9] supports cleanup whereas multivariant execution does not.

Table 1 shows a list of representative dynamic platform techniques and their features.

We use one of the above techniques, Talent, to quantitatively analyze the effectiveness of dynamic platforms. Although Talent does not natively support multi-instance and cleanup, we augment it with these features to understand their impact. The main reason for using Talent was its code availability, but we show that our analysis can be generalized based on the features of the techniques.

In this work, our goal is not to provide arguments for merits or demerits of any of the proposed dynamic platform techniques. Rather, we strive to quantitatively evaluate dynamic platforms as a cyber defense mechanism and study various features that can significantly change their impact.

2.1 Talent

Talent [7] is a technique that allows live migration of applications across diverse platforms. It uses operating-system-level virtualization (OpenVZ [20]) to sandbox an application and migrate the environment. For internal process state migration, Talent uses a portable checkpoint compiler (CPPC [21]) to insert checkpointing instructions into a code. At the time of migration, it pauses a process, checkpoints its state, moves the state to the next platform, and resumes the execution. Some portions of the code are re-executed in order to construct the entire state.

Since it allows an application to run on different operating systems and architecture, Talent provides diversity. Also, it is a limited duration technique, because it can pause a process and resume it on a different platform. However, it does not natively support multi-instance since one platform is active at a time; it does not implement cleanup either.

Talent has been implemented on Intel Xeon 32-bit, Intel Core 2 Quad 64-bit, and AMD Opteron 64-bit processors. It has also been tested with Gentoo, Fedora (9, 10, 11, 12, and 17), CentOS (4, 5, and 6.3), Debian (4, 5, and 6), Ubuntu (8 and 9), SUSE (10 and 11), and FreeBSD 9 operating systems.

3 Threat Model

We discuss multiple threat models in this paper but analysis shows that they share common features. To make the analysis more precise, we explicitly describe the core threat model in this section. Variations upon the core threat model are described in the other sections as appropriate.

In our model, the defender has a number of different platforms to run a critical application. The attacker has a set of exploits (attacks) that are applicable against some of these platforms, but not the others. We call the platforms for which the attacker has an exploit "vulnerable" and the others "invulnerable." In a strict systems security terminology, vulnerable does not imply exploitable; without loss of generality, we only consider exploitable vulnerabilities. An alternative interpretation of this threat model is that the vulnerabilities are exploitable on some platforms, but not on the other ones.

The defender does not know which platforms are vulnerable and which are invulnerable, nor does she have detection capabilities for the deployed exploits. This scenario, for example, describes the use of zero-day exploits by attackers, for which no detection mechanism exists by definition.

Since there is little attempt to isolate the inactive platforms in dynamic platform systems, we assume that all platforms are accessible by the attacker, and the attacker attempts to exploit each one.

The attacker's goal is what creates the variations in our threat model. For example, one success criteria may be for the attacker to compromise the system for a given period of time to cause irreversible damage (e.g. crash a satellite), while a different success criteria gives the attacker gradual gain the longer the system is compromised (e.g. exfiltration of information). Different techniques with different features provide varying protections against these goals which we study in the subsequent sections.

4 Experiments

4.1 Experiment Setup

To understand the protection provided by dynamic platforms, we start by performing simple experiments with Talent and two real-world exploits. We observe that contrary to the naïve view, even these simple experiments result in very complex results which highlight a number of subtleties about dynamic platforms.

To perform the experiments, a notional application with C back-end and GUI front-end has been ported to Talent. The application's back-end performs attestation of machines within a local network and its front-end displays the result. However, the details of the application are unimportant for the evaluations done in this work, so for the sake of brevity we do not discuss them here.

On the testbed, we have a pool of five different platforms: Fedora on x86, Gentoo on x86, Debian on x86_64, FreeBSD on x86, and CentOS on x86. The application runs for a random amount of time on a platform before migrating to a different one (i.e. platform duration).

The attacker's goal in the experiments is to control the active platform for some time T. Since in a real scenario the vulnerability of the platform is unknown, we may consecutively migrate to multiple vulnerable platforms, in which case the attacker wins. To implement this scenario on the testbed, we launch two real exploits against Talent. The first is the TCP MAXSEG exploit which triggers a divide by zero vulnerability in `net/ ipv4/tcp.c` (CVE-2010-4165) to perform a DoS attack on the platform. Only the Gentoo platform is vulnerable to this attack. The second attack is the Socket Pairs exploit which triggers a garbage collection vulnerability in `net/unix/ garbage.c` (CVE-2010-4249) to saturates the CPU usage and file descriptors. The Fedora and CentOS platforms are vulnerable to this attack. Our Debian and FreeBSD platforms are not vulnerable to these exploits.

In each configuration, we select $N \in (1, 5)$ platforms. For each trial, the application randomly migrates across those N platforms without immediate repeat. In the case of $N = 1$ (baseline), the application remains on the same platform during the entire trial. Without loss of generality, the duration on each platform (d) is chosen randomly and uniformly from $40 - 60$ seconds. Although we have no reason to believe that these are the appropriate values for a real-world application, we will show later that the actual values of the duration (d) and attacker's goal (T) are inconsequential to our experiments and can be parametrized.

One or both exploits become available to the attacker at random times during each trial. As a result, zero to three platforms can be compromised (zero when the exploit is not effective against the set of platforms and three when both exploits are available and Fedora, CentOS, and Gentoo are in the pool of platforms). When the exploit is launched, its payload reaches all of the platforms in the selected set at once (not one after another). This approach tries to model the behavior of network-based exploits that propagate to all machines within a network very rapidly. Each trial runs for 15 minutes. We collect 300 trials for each configuration. We also collect a central log which includes a timestamp, the status of each platform (up or down), and the active platform and a local log (for verification purposes) which also includes finer-grained CPU load for each platform.

Fig. 1 illustrates one trial with 3 platforms. The red arrows show when exploits are launched. In this case, platforms 2 and 5 are vulnerable to exploits 1 and 2 respectively. A shaded rectangle shows a compromised platform while a white rectangle shows an uncompromised one (vulnerable or invulnerable).

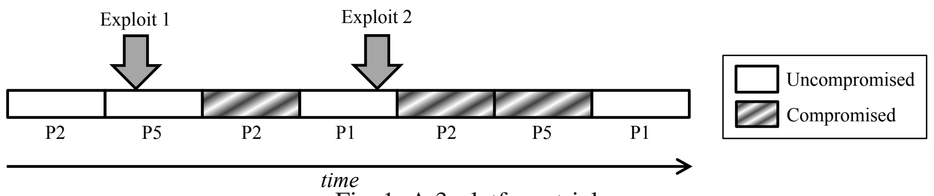


Fig. 1: A 3-platform trial

4.2 Experiment Results

We calculate the value of the metric, which is the percentage of time that the attacker is in control for longer than T and present these results in Fig. 2.

The results are completely perplexing. In fact, the results are so counter-intuitive that we initially thought that some mistakes have been made in collecting them. We can at least observe the following peculiarities in the results.

- The 1-platform result is very different than the others and seems to estimate a straight line for $T > 100$ sec.
- More platforms does not always result in lower chance of attacker success. Specifically for $60 < T < 120$, more platforms result in higher chance of success for the attacker.
- There are several downward steps in the curves for more than one platform at $T = 60, 120, 180, \ldots$.
- For $T > 120$, more platforms result in lower chance of attacker success and that remains the case for larger values of T.

The complexity of the results suggest that various effects should be in play which we explain one by one in the next section.

5 Abstract Analysis

Much of the analysis of one system using dynamic platforms as a defense applies to any such system. First, we explain the effects that contribute to our experiment results and then we generalize our analysis to any dynamic platform technique.

5.1 Limited Duration Effect

The first effect contributing to the results is the limited duration effect. Let d be the duration of the transaction on a platform, T be the period that the attacker must be

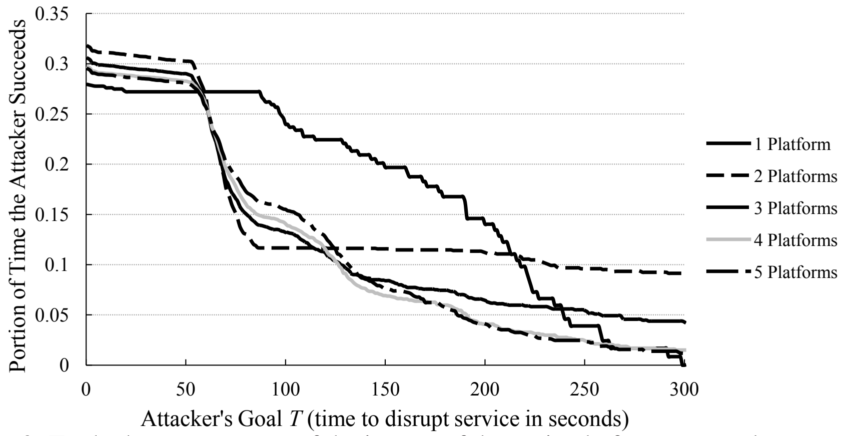


Fig. 2: Testbed measurements of the impact of dynamic platform on attacker success

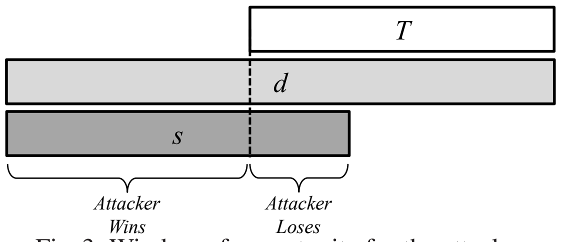


Fig. 3: Window of opportunity for the attacker

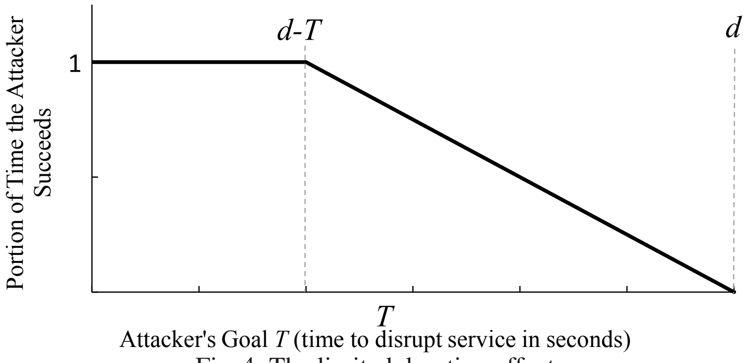


Fig. 4: The limited duration effect

present, and s the start time of attack. If $T > d$, the attacker can never win. For $T < d$, the attacker can only win if she starts early enough during the $d-T$ interval. As a result, the probability of winning for the attacker is a decreasing linear function of T.

Then the probability that the attack succeeds is given by

$$Pr_{success} = \min\left(1, \max\left(0, \frac{d-T}{s}\right)\right)$$

This explains the general decreasing trend for the probability of success as a function of attacker's goal in Fig. 2.

Counter-intuitively, this effect also explains the straight-line result for the 1-platform experiment in Fig. 2. Although in the 1-platform case, the platform never changes, the probability of success decreases linearly with time because the entire trial has a limited duration. The attacker cannot possibly win if she starts late even if that single platform is vulnerable. This explains the similarity of the 1-platform result in Fig. 2 and Fig. 4.

5.2 Diversity Effect

Informally speaking, the intuition behind the concept of diversity is that it is harder for an attacker to compromise different platforms than it is to compromise homogeneous ones. Since we assume that the platforms are all available and no separation exists between them, in the case of homogeneous platforms, they can all be compromised by an exploit that works against one of them. On the other hand, if the platforms are diverse (which is the case in our experiments), an exploit can work against some of them, but not the other ones.

In practice, diversity creates an effect which occurs when the required attacker goal T passes between various multiples of the duration of each platform d. For example, if the attacker goal passes from being a bit less than a single platform duration to a bit more, then instead of a single vulnerable platform, two need to be used consecutively. The same effect happens as we transition from two to three and so on. The result is downward steps in the curve when the required attacker goal passes multiples of the platform duration. Fig. 5 illustrates this impact when three out of five platforms are vulnerable. The first platform is trivially vulnerable with $\frac{3}{5}$ probability. Since we do not have immediate repeats, the subsequent platforms are chosen from the four remaining ones of which two are vulnerable, so the probability that the second platform is vulnerable if the first one is vulnerable is $\frac{2}{4}$. As a result, both the first and the second platforms are vulnerable with probability $\frac{3}{5} \times \frac{2}{4}$. If we extend the analysis, the first, second, and third platforms are vulnerable with probability $\frac{3}{5} \times (\frac{2}{4})^2$, and so on (see Fig. 5).

5.3 Multi-Instance Effect

In the multi-instance case, the system is compromised if the majority of platforms are compromised. Although Talent does not natively support multi-instance, we augment it with a simple voting mechanism to analyze the impact of running multiple platforms simultaneously. With the same experiment setup as described in section 4 we analyze the probability of success when the application runs on multiple platforms and the majority of the platforms are compromised.

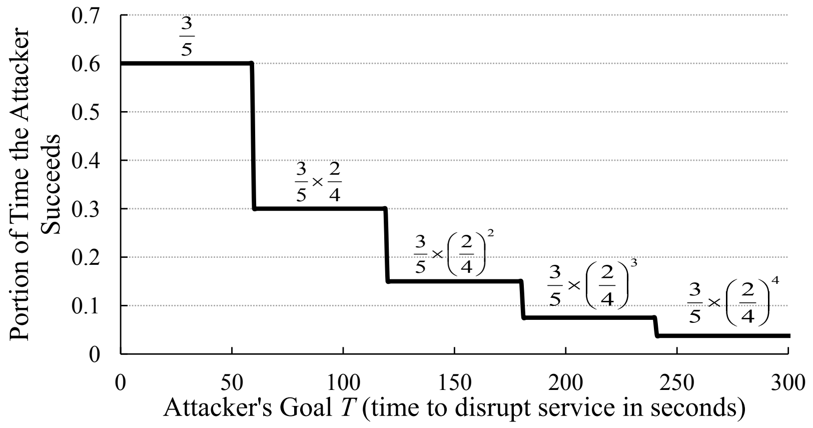


Fig. 5: The diversity effect

When the system is multi-instance, if there is no platform change, the case becomes trivial; that is, if the majority of the platforms are not vulnerable, the system as a whole is never compromised and the attacker never wins. On the other hand, if the majority of the platforms are vulnerable, the attacker wins as soon as the exploits are launched and remains in control indefinitely. As a result, we only show the effect when the platform change happens. Moreover, the 1-platform and 5-platform cases are also trivial, so we only show the results for a 3-platform setup. In this setup, the application runs on three platforms simultaneously. For each platform, the application migrates to a new platform uniformly randomly after spending 40-60 seconds. Thus, the migrations may be out-of-sync, but at each instance of time the application is running on three diverse platforms.

The multi-instance effect is shown in Fig. 6. The single instance result is the same as the 3-platform setup in Fig. 2.

Counter-intuitively, the multi-instance setup is less secure for small values of T. This arises from a combinatorial effect. Since three of the five platforms are vulnerable, there are three configurations in which the majority is not vulnerable (the two invulnerable platforms selected with one other vulnerable platform) which is expressed by $C\left(3,1\right)$ where $C\left(x,y\right)=\frac{x!}{y!(x-y)!}$ is the combinatorial choice function. The total number of choices is $C\left(5,3\right)=10$. As a result, the defender wins with the probability of 30% and thus, the attacker wins with the probability of 70%. This is why the multi-instance case starts from 0.7. With the single instance case this probability is smaller because there is a higher probability of a combination with an invulnerable platform. In other words, when the majority of the platforms are vulnerable (3 out of 5 in this case), there is a higher probability that if we choose three platforms, two or more of them are vulnerable $(1-C\left(3,1\right))$ than if we choose just one platform and that is vulnerable $(\frac{3}{5})$. We will explain this effect in more details in Section 6.1.

5.4 Cleanup Effect

A dynamic platform system supports cleanup if every inactive platform in restored into its pristine state. Talent does not natively support cleanup either, but we augment it

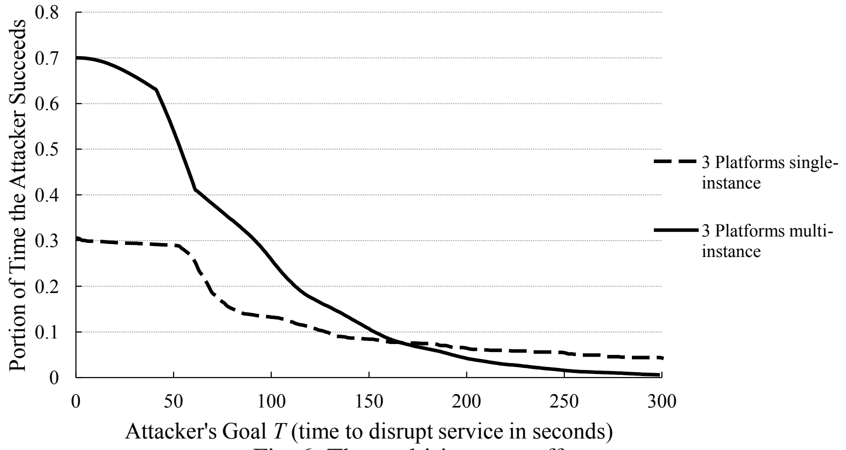


Fig. 6: The multi-instance effect

with a cleanup capability to evaluate its impact. As discussed earlier, techniques such as SCIT [9] and MAS [18] implement cleanup.

The impact of cleanup is trivial if the exploit is only launched once and never repeated; the attacker may compromise the active platform for the remainder of the time on that platform, but when the platform changes, the system becomes pristine and the attacker never wins again. This is because the inactive platforms are being cleaned while the attacker attacks the active one. Consequently, in the case of a non-repeating exploit, the portion of time the attacker is in control amortizes with the duration of the trial.

Here, we evaluate the non-trivial case where the exploit is repeated frequently. We re-launch the exploit with mean time between attacks (MTBA) set at 20, 40, and 60 seconds. Fig. 7 illustrates the impact of cleanup. As can be observed, for any attacker goal of greater than 60 seconds, the chance of success for the attacker drops to zero. This makes sense because the inactive platforms are restored to their pristine state, so the application can never migrate to an already compromised platform. As a result, the attacker can only win if her goal is shorter than the maximum duration of time on a single platform, which is 60 seconds.

As the results suggest, cleanup can greatly improve the protection offered by dynamic platform techniques since it significantly reduce the window of opportunity for an attacker. It is advisable that all dynamic platform techniques should support cleanup.

5.5 Smoothing Effects

A few effects contribute to the smoothness of the edges of the curves depicted in Fig. 2. For example, the downward steps are not sharp transitions (similar to a step function). Rather, they are smoother curvatures. For the sake of completeness, we explain a few factors that contribute to this smoothness.

First, the time spent on a platform is not fixed; rather, it is a random variable uniformly selected between 40 and 60 seconds. This is an important smoothing factor because it makes the time on a platform non-deterministic and as a result, it makes the threshold for passing between multiples of the platforms also smooth.

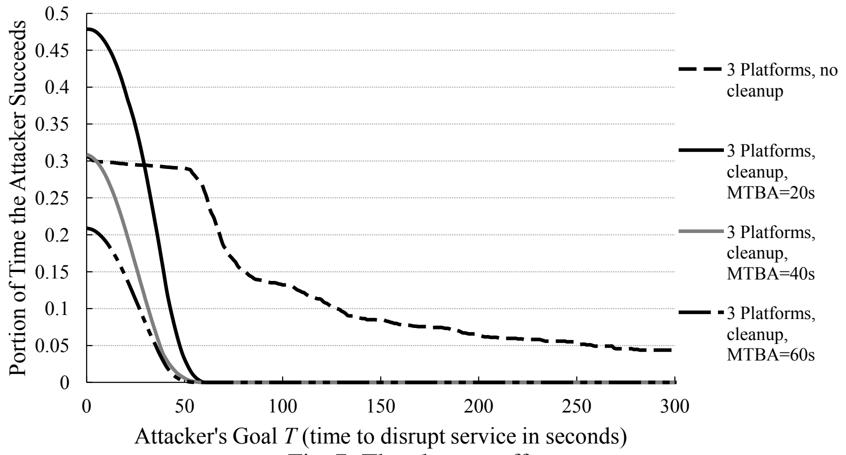


Fig. 7: The cleanup effect

Second, the exploits are also launched at random times instead of the beginning of the trial. This factor is not crucial in evaluating dynamic platforms and its only tangible impact is making the curves smoother.

Third, we assumed that as soon as the exploit is launched the vulnerable platforms are compromised. In reality, the time it takes for the exploit to successfully compromise a platform after reaching it is non-zero which also makes the results smoother. For example, the Socket Pairs exploit used in the experiments takes a few seconds to saturate the file descriptors.

Fourth, networking, OS scheduling, and various other delays also make the results smoother and in some cases noisier.

6 Generalized Model of Dynamic Platform Techniques

In this section, we use the knowledge of our experiments and the effects that we explained in the previous section to develop a generalized model of the dynamic platform techniques.

We can categorize the problem space according to a number of properties:

- The attackers control requirement can either be aggregate or continuous. In the aggregate case, any period of time during which the attacker controls a platform counts and *aggregates* towards the payoff. Data exfiltration attacks are an example of attacks that require aggregate control. In the continuous case, only the time since the most recent compromise during which the attacker has *continuous* control of the platform counts towards the payoff. For example, attacks that leak crypto keys through remote side channel attacks require continuous control since that key may only be valid for the most recent session.
- The attackers payoff can be either fractional or binary (all or nothing). In the fractional case, the attacker is rewarded more, the longer she controls the platform. Data exfiltration attacks are an example of fractional payoff. In the binary case, the

α	Number of vulnerable platforms
β	Number of invulnerable platforms
p^k	Platform at migration step k
$v\left(p^k\right)$	Platform at migration step k is vulnerable
$\neg v\left(p^k\right)$	Platform at migration step k is not vulnerable
$Pr\left(v\left(p^k\right)\right)$	Probability that $v\left(p^k\right)$
Pr_{vv}	$P\left(v\left(p^{k+1}\right)\vert v\left(p^k\right)\right)$
Pr_{ii}	$P\left(\neg v\left(p^{k+1}\right)\vert \neg v\left(p^k\right)\right)$

Table 2: Notation describing dynamic platform system

attacker is not rewarded before a known period of control, and then she is fully rewarded at once. Attacks on critical infrastructure systems to cause a physical impact (e.g. to cause a blackout) are an example of binary payoff.
– The platform change model can include random with repeat, random without repeat, and periodic permutation.

We will define the abstract model of a dynamic platform system $\mathcal{P}$ as a system that migrates through a finite fixed collection of platforms $\{p_i\}$. Each platform either has or does not have a property exploitable by the attacker which we call vulnerable. In the first approximation to the model we assume that the platforms are fully independent. We will use the notation presented in Table 2.

6.1 Attacker Aggregate Control

When the attacker requires only aggregate control, there are two main subcategories according to the attacker's payoff. The fractional case is trivially determined by the ratio of α and β. In the binary case, wherein the attacker wins by controlling a specified fraction of the vulnerable time, the defender may optimize via an initial subselection of platforms in a process reminiscent of gerrymandering. For example, if $\alpha = 3$ and $\beta = 2$ and the attacker wants to control greater than 50% of the time, then the defender should simply expect to lose should all platforms be utilized. By contrast if the defender randomly subselects two platforms then the defender can reduce the attacker's expectation of winning to

$$\frac{C\left(3,2\right)}{C\left(5,2\right)} = \frac{3}{10} = 30\%,$$

where $C\left(x,y\right) = \frac{x!}{y!(x-y)!}$ is the combinatorial choice function. Here the value of 2 as the number of platforms chosen.

Generally, if t is the percentage of time that the attacker requires for success and we subselect j platforms from the total $\alpha + \beta$, then the probability of attacker success is

$$Pr_{success} = \sum_{i=\lceil t\cdot j\rceil}^{\min(\alpha,\, j)} \frac{C\left(\alpha,\, i\right)\cdot C\left(\beta,\, j-i\right)}{C\left(\alpha+\beta,\, j\right)},$$

in the steady-state model.

Repeat	Vuln	¬Vuln	$Pr\left(v\left(p^{k+1}\right)\right)$	$Pr\left(v\left(p^{k+1}\right)\mid v\left(p^{k}\right)\right)$	$Pr\left(v\left(p^{k+j}\right)\mid v\left(p^{k+j-1}\right)\&\ldots\&v\left(p^{k}\right)\right)$
Without	α	β	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha-1}{\alpha+\beta-1}$	$\frac{\alpha-j}{\alpha+\beta-j}$
With	α	β	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha}{\alpha+\beta}$

Table 3: Conditional Probabilities

6.2 Attacker Continuous Control

When the attacker requires continuous control, the defender can use the subselection strategy as above as well as leveraging conditional probabilities. These conditional probabilities are given in Table 3.

Here, we observe that $\frac{\alpha}{\alpha+\beta} > \frac{\alpha-j}{\alpha+\beta-j}$ so long as β and j are both greater than zero. As such, migrating *without* immediate repeat, while not influencing the fraction of vulnerable platforms selected, tends to reduce successful sequences for the attacker. We note that the influence is greater when a smaller number of platforms is used. Our later experiment will use 3 vulnerable and 2 invulnerable platforms which is a sufficiently small number to have a strong influence upon the conditional probabilities.

This reduces to the Markov chain:

$$\begin{tikzcd}
 & V \arrow[loop right, "Pr_{vv}"] \arrow[dd, bend left, "1-Pr_{vv}"] \\
\boxed{\text{start}} \arrow[ur, "Pr_v"] \arrow[dr, "Pr^k=1-Pr_v"'] & \\
 & I \arrow[loop right, "Pr_{ii}"] \arrow[uu, bend left, "1-Pr_{ii}"]
\end{tikzcd}$$

6.3 Attacker Fractional Payoff Model

The steady state of attacker control of the system can be modeled using Markov chains with states I and V referring to invulnerable and vulnerable respectively. While the simple Markov model describing the transitions $\{I,V\} \longrightarrow \{I,V\}$ describes the base behavior of the system, it does not naturally capture the notion of repeated vulnerable states. We can adapt this chain to one with a richer collection of states

$$\left\{I, IV, IV^2, \ldots, IV^{n-1}, V^n\right\} \longrightarrow \left\{I, IV, IV^2, \ldots, IV^{n-1}, V^n\right\}$$

which support runs of length n. The probability of invulnerable to invulnerable transition is given by

$$Pr_{ii} = Pr\left(\neg v\left(p^{k+1}\right)\mid \neg v\left(p^{k}\right)\right) = \frac{\beta-1}{\alpha+\beta-1}$$

and the probability of vulnerable to vulnerable transition is given by

$$Pr_{vv} = Pr\left(v\left(p^{k+1}\right)\mid v\left(p^{k}\right)\right) = \frac{\alpha-1}{\alpha+\beta-1}$$

The Markov model looks like

$$\begin{tikzcd}
 & IV \arrow[r, "Pr_{vv}"] \arrow[dl, bend left, "1-Pr_{vv}"] & IV^2 \arrow[dll, bend left, "1-Pr_{vv}"'] \arrow[dr, dotted] & \\
I \arrow[loop below, "Pr_{ii}"] \arrow[ur, bend left, "1-Pr_{\neg v}"] & & & IV^{n-1} \arrow[lll, "1-Pr_{vv}"'] \arrow[dl, "Pr_{vv}"] \\
 & & V^n \arrow[ull, bend right, "1-Pr_v"'] \arrow[loop below, "Pr_{vv}"] &
\end{tikzcd}$$

which has the $(n+1) \times (n+1)$ Markov transition matrix is given by

$$\begin{bmatrix}
Pr_{ii} & 1-Pr_{vv} & & & & & \\
1-Pr_{vv} & & Pr_{vv} & & & & \\
1-Pr_{vv} & & & Pr_{vv} & & & \\
1-Pr_{vv} & & & & Pr_{vv} & & \\
1-Pr_{vv} & & & & & \ddots & \\
1-Pr_{vv} & & & & & & Pr_{vv} \\
1-Pr_{vv} & & & & & & Pr_{vv}
\end{bmatrix}.$$

This transition matrix has the steady state eigen-vector

$$\left[\tfrac{\beta}{\alpha+\beta} \quad a_v \cdot Pr_{vv} \quad a_v \cdot Pr_{vv}^2 \quad \cdots \quad a_v \cdot Pr_{vv}^{n-1} \quad a_v \cdot \textstyle\sum_{i=n}^{\infty} Pr_{vv}^i \right]$$

where

$$a_v = \frac{\alpha}{\alpha+\beta} \cdot \left(\frac{1 - Pr_{vv}}{Pr_{vv}} \right).$$

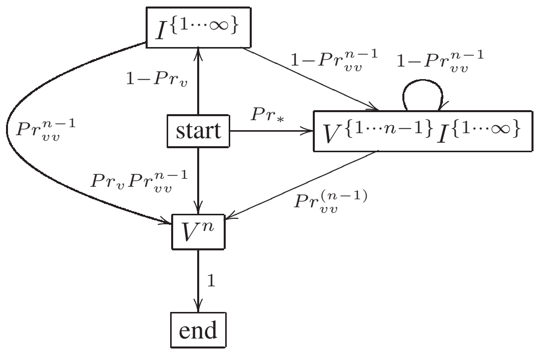
This can be used to compute the steady state behavior of the system. If the attacker success begins after n steps then the steady state is given by the right most term in the eigen vector $a_v \cdot \sum_{i=n}^{\infty} P_v^i = \frac{\alpha}{\alpha+\beta} - a_v \cdot \sum_{i=0}^{n-1} P_v^i$. If the attacker success includes the steps leading to a run of n steps then we must also include vulnerable states weighted by the probability that they will become a run of n vulnerable states and the contribution to the run: the probability that IV^{n-1} will become $V^n$is Pr_V, the probability that IV^{n-2} will become $V^n$is $2{\cdot}Pr_V^2$ and so forth. Reducing that equation, we find that the expected period of attacker control $L(n)$ is

$$L(n) = 1 - \frac{(1 - Pr_{\neg v})^{-1} + (1 - Pr_v) \sum_{i=0}^{n-1} i \cdot Pr_v^{i-1}}{(1 - Pr_{\neg v})^{-1} + (1 - Pr_v)^{-1}}$$

which is one minus the percentage of time that the defender is in control.

6.4 Attacker Binary Payoff Model

In the binary payoff model with random selection (with or without immediate repeats), the attacker will eventually win so long as it is combinatorially possible in the same manner that a person flipping a coin will eventually observe a sequence of ten, or ninety-two, heads in a row. Here metrics might reasonably be based in the mean time until attacker victory. These can be analyzed in a fashion similar to the steady state model:



where $Pr_* = Pr_v\left(1 - Pr_{vv}^{n-1}\right)$. We can use this to evaluate the expected time $L'(n)$ to attack compromise as the probabilistically weighted sum of all path lengths

$$\begin{aligned} L'(n) =& n + \frac{1-Pr_v}{1-Pr_{ii}} + \\ & \left(Pr_{vv}^{1-n} - 1\right) \cdot \\ & \left(\frac{1 - n \cdot Pr_{vv}^{n-1} + (n-1) \cdot Pr_{vv}^{n}}{\left(1 - Pr_{vv}^{n-1}\right) \cdot \left(1 - Pr_{vv}\right)} + \frac{1}{1-Pr_{ii}}\right) \end{aligned} . \quad (1)$$

Hence, in scenarios such as ‘crash the satellite’, Eq. (1) computes the expected time before the adversary is able to take down the service.

7 Simulation Results

In order to verify that we have captured the major effects in our analysis and that our generalized model of dynamic platforms is valid, we simulate the Markov chain that corresponds to our testbed experiments. Our testbed experiments assumed migration with no immediate repeat, continuous control, and fractional payoff which is modeled using the Markov chain in section 6.3. We run a Monte Carlo simulation on that model with the same parameters as our testbed experiments: $40 - 60$ second time of each platform, three vulnerable platforms out of five total, exploits launched at random times during each trial, and each trial runs for 15 minutes. The results are presented in Fig. 8. In the figure, the testbed measurements are also overlaid on the simulated results using dotted lines for comparison.

As can be observed, the simulation results match the testbed measurements very closely. This validates the fact that we have indeed captured at least the major effects that contribute to the effectiveness of dynamic platform techniques. Note that the smoothing effects (e.g. random duration on a platform and random exploit launch times) are captured in the simulation results since we have captured them in the model. However, various jitters and delays (e.g. networking, OS scheduling, etc.) are not in the

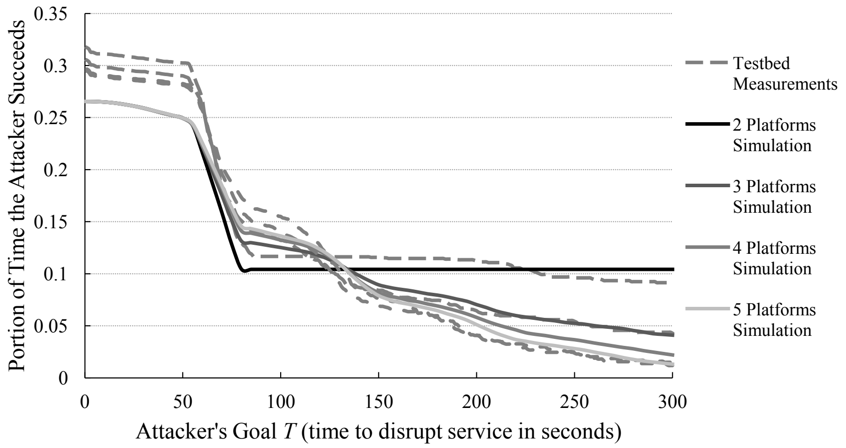


Fig. 8: Simulation results from the generalized model. The testbed measurements are also shown in dotted lines

model which can explain the small amount of discrepancy between the simulated and measured results. Table 4 shows the mean squared error (MSE) of the simulation results compared to the testbed measurements.

7.1 Discussion

One important observation to be made for both the simulated and measured results is that for small attacker goals (T), fewer platforms actually perform better. This is due to the fact that in situations where the attacker wins quickly, more platforms present a larger attack surface. As a result, the attacker wins if she can compromise any of the platforms. In other words,

$\frac{T}{d} \to 0$: Attacker wins iff *any* platform is vulnerable

The value of dynamic platforms can only be observed for attacker goals that are large with respect to the duration of each platform ($T \gg d$). This is an important parameter when deploying dynamic platform systems; the duration of each platform must be selected short enough based on the service requirements of the system. For example, if the system has to survive and provide service within 5 minutes (i.e. the attacker goal is disrupting service longer than $T = 5$ minutes), the platform duration must be $d << 5$ min. In other words,

$\frac{T}{d} \to \infty$: Attacker wins iff *all* platforms are vulnerable

Note that there may be practical considerations when choosing small platform duration. If the platform changes too rapidly (i.e. very small d), it can disrupt the normal mission of the system.

Number of Platforms	Mean Squared Error
2 Platforms	634×10^{-6}
3 Platforms	329×10^{-6}
4 Platforms	322×10^{-6}
5 Platforms	257×10^{-6}

Table 4: Mean squared error of the simulated model compared to the testbed measurements

8 Lessons Learned

Our work in analyzing dynamic platform techniques has provided five main lessons.

The first is that many effects contribute to a dynamic platform system. Although these systems have been proposed in many different forms in the literature, little work has been done to identify and quantify these effects which can be very counter-intuitive. On the other hand, when these effects are studied and understood, even first-order models can closely estimate the system behavior.

The second is that experiments such as ours using real-world technologies on a testbed can shed light on some of the complex dynamics of active systems and can be used as a way to identify and quantify the major contributing effects of such systems.

The third is that threat models are crucial in understanding the protection provided by a defensive technique and they are also instrumental in quantitatively measuring such protections. As can be observed in our results, while a technique can provide significant protection against one type of threat (e.g. long-duration attacks that can have fractional gain for the attacker such as slow data exfiltration), it may actually degrade the security of the system for another one (e.g. short duration attacks causing an irreversible impact). In fact, threat models should be an integral part of metrics and measurements of effectiveness [22].

The fourth is that testbed experiments, abstract analysis, and modeling and simulation can be used together to perform quantitative evaluation of defensive techniques in general. These different approaches can identify subtle effects and dynamics. Moreover, they can provide the verification and validation necessary to ensure that the results are indeed correct.

The final lesson is that some features of the proposed techniques, such as cleanup, can significantly reduce the likelihood of success for attacks. When designing new techniques, quantitative evaluations such as what we have done in this paper can be used to decide the important features to support in order to provide the most protection with the least performance overhead.

9 Related Work

Various dynamic platform techniques have been proposed in the literature. As mentioned earlier, The Self-Cleansing Intrusion Tolerance (SCIT) project rotates virtual machines to reduce the exposure time. SCIT-web server [23] and SCIT-DNS [24] preserve the session information and DNS master file and keys, respectively, but not the internal state of the application. The Resilient Web Service (RWS) Project [25] uses a virtualization-based web server system that detects intrusions and periodically restores them to a pristine state. Certain forms of server rotation have been proposed by Blackmon and Nguyen [26] and by Rabbat *et al.* [27] in an attempt to achieve high availability servers.

High-level forms of temporal platform changes have been proposed by Petkac and Badger [28] and Min and Choic [19] to build intrusion tolerant systems although the diversification strategy is not as detailed in these efforts. Compiler-based multivariant [3–5, 15, 29] and N-variant systems [16] propose another way of achieving platform diversity. Holland *et al.* propose diversifying machine descriptions using a virtualization layer [6]. A similar approach with more specific diversification strategy based on instruction sets and calling sequences has been proposed by Williams *et al.* [2]. Wong and Lee [30] use randomization in the processor to combat side-channel attacks on caches.

On the evaluation side, Manadhata and Wind [31] propose a formal model for measuring a system's attack surface that can be used to compare different platforms. Evans *et al.* [32] develop models to measure the effectiveness of diversity-based moving target technique. They evaluate the probability of attack success given the time duration of attack probing, construction, and launch cycles and the entropy of randomness in the target system. They evaluate the impact of various attacks on moving target systems including circumvention, deputy, brute force, entropy reduction, probing, and incremental attacks.

There has been numerous modeling attempts in the literature for diversity systems or N-version programming such as those done by Popov and Mladenov [33], or Arlat *et al.* [34]. However, they focus on accidental faults, not malicious attacks.

10 Conclusion

In this paper, we have quantitatively studied cyber defenses based on dynamic platform techniques. We used testbed experiments to collect results from an actual technique. The unexpected and complex results motivated us to perform an abstract analysis to explain the various effects that contribute to the protection. We extended our analyses to the main features provided by the dynamic platforms proposed in the literature. Based on these effects, we then developed a generalized model of dynamic platforms. In order to ensure that we have captured the major effects, and to verify the model and validate our testbed results, we simulated the same sets of experiments using the generalized model. The closely matching results enhance the confidence in the results and validate the fact that we have at least captured the main effects.

Our results suggest that while dynamic platforms are useful for mitigating some attacks, it is of critical importance to understand the threat model one aims to defend against. While dynamic platforms can be effective against long-period attacks with gradual gains (e.g. data exfiltration), they can be detrimental for short-period attacks with instantaneous gains (e.g. a malware causing an irreversible impact in a control system).

The future work in this domain will focus on performing more experiments with such systems, extending the analysis to other dynamic platform techniques and other randomization and diversity approaches, and analyzing the second order behavior such as adaptive adversaries who change tactics based on the deployed defenses.

11 Acknowledgement

We would like to thank Charles Wright, Mark Rabe, Paula Donovan, and William Streilein for their insights and contributions to this work.

References

[1] Networking, F., Research, I.T., (NITRD), D.: Federal Cybersecurity Game-change R&D Themes (2012) http://cybersecurity.nitrd.gov/page/federal-cybersecurity-1.

[2] Williams, D., Hu, W., Davidson, J.W., Hiser, J.D., Knight, J.C., Nguyen-Tuong, A.: Security through diversity: Leveraging virtual machine technology. IEEE Security and Privacy **7**(1) (January 2009) 26–33

[3] Salamat, B., Jackson, T., Wagner, G., Wimmer, C., Franz, M.: Runtime defense against code injection attacks using replicated execution. Dependable and Secure Computing, IEEE Transactions on **8**(4) (july-aug. 2011) 588 –601

[4] Salamat, B., Gal, A., Jackson, T., Manivannan, K., Wagner, G., Franz, M.: Multi-variant program execution: Using multi-core systems to defuse buffer-overflow vulnerabilities. In: International Conference on Complex, Intelligent and Software Intensive Systems. (2008)

[5] Jackson, T., Salamat, B., Wagner, G., Wimmer, C., Franz, M.: On the effectiveness of multi-variant program execution for vulnerability detection and prevention. In: Proceedings of the 6th International Workshop on Security Measurements and Metrics. (2010) 7:1–7:8

[6] Holland, D.A., Lim, A.T., Seltzer, M.I.: An architecture a day keeps the hacker away. SIGARCH Comput. Archit. News **33**(1) (March 2005) 34–41

[7] Okhravi, H., Comella, A., Robinson, E., Haines, J.: Creating a cyber moving target for critical infrastructure applications using platform diversity. International Journal of Critical Infrastructure Protection **5**(1) (2012) 30 – 39

[8] Saidane, A., Nicomette, V., Deswarte, Y.: The design of a generic intrusion-tolerant architecture for web servers. Dependable and Secure Computing, IEEE Transactions on **6**(1) (jan.-march 2009) 45 –58

[9] Bangalore, A., Sood, A.: Securing web servers using self cleansing intrusion tolerance (scit). In: Second International Conference on Dependability. (2009) 60 –65

[10] Huang, Y., Arsenault, D., Sood, A.: Incorruptible system self-cleansing for intrusion tolerance. In: Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International. (april 2006) 4 pp. –496

[11] Arsenault, D., Sood, A., Huang, Y.: Secure, resilient computing clusters: Self-cleansing intrusion tolerance with hardware enforced security (scit/hes). In: Proceedings of the The Second International Conference on Availability, Reliability and Security. ARES '07, Washington, DC, USA, IEEE Computer Society (2007) 343–350

[12] Okhravi, H., Hobson, T., Bigelow, D., Streilein, W.: Finding Focus in the Blur of Moving-Target Techniques. IEEE Security & Privacy (Mar/Apr 2014)

[13] Scott, K., Davidson, J.: Strata: A Software Dynamic Translation Infrastructure. Technical Report CS-2001-17 (2001)

[14] Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. In: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation. PLDI '07, New York, NY, USA, ACM (2007) 89–100

[15] Salamat, B., Gal, A., Franz, M.: Reverse stack execution in a multi-variant execution environment. In: In Workshop on Compiler and Architectural Techniques for Application Reliability and Security. (2008)

[16] Cox, B., Evans, D., Filipi, A., Rowanhill, J., Hu, W., Davidson, J., Knight, J., Nguyen-Tuong, A., Hiser, J.: N-variant systems: a secretless framework for security through diversity. In: Proceedings of the 15th conference on USENIX Security Symposium. (2006)

[17] Crouse, M., Fulp, E.: A moving target environment for computer configurations using genetic algorithms. In: Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium on. (Oct 2011) 1–7

[18] Huang, Y., Ghosh, A.K.: Introducing diversity and uncertainty to create moving attack surfaces for web services. In: Moving Target Defense. (2011) 131–151

[19] Min, B.J., Choi, J.S.: An approach to intrusion tolerance for mission-critical services using adaptability and diverse replication. Future Gener. Comput. Syst. (2004) 303–313

[20] Kolyshkin, K.: Virtualization in linux. White paper, OpenVZ (September 2006)

[21] Rodríguez, G., Martín, M.J., González, P., Touriño, J., Doallo, R.: Cppc: a compiler-assisted tool for portable checkpointing of message-passing applications. Concurr. Comput. : Pract. Exper. **22**(6) (April 2010) 749–766

[22] R.P. Lippmann, J.F. Riordan, T.H. Yu, K.K. Watson: Continuous Security Metrics for Prevalent Network Threats: Introduction and First Four Metrics. Technical report, MIT Lincoln Laboratory (May 2012)

[23] Bangalore, A.K., Sood, A.K.: Securing web servers using self cleansing intrusion tolerance (scit). In: Proceedings of the 2009 Second International Conference on Dependability. (2009) 60–65

[24] Yih Huang, David Arsenault, A.S.: Incorruptible self-cleansing intrusion tolerance and its application to dns security. AJournal of Networks **1**(5) (September/October 2006) 21–30

[25] Huang, Y., Ghosh, A.: Automating intrusion response via virtualization for realizing uninterruptible web services. In: Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on. (july 2009) 114 –117

[26] Blackmon, S., Nguyen, J.: High-availability file server with heartbeat. System Admin, The Journal for UNIX and Linux Systems Administration **10**(9) (2001)

[27] Rabbat, R., McNeal, T., Burke, T.: A high-availability clustering architecture with data integrity guarantees. In: IEEE International Conference on Cluster Computing. (2001)

[28] Petkac, M., Badger, L.: Security agility in response to intrusion detection. In: in 16th Annual Computer Security Applications Conference (ACSAC. (2000) 11

[29] Jackson, T., Salamat, B., Homescu, A., Manivannan, K., Wagner, G., Gal, A., Brunthaler, S., Wimmer, C., Franz, M.: Compiler-generated software diversity. In: Moving Target Defense. (2011) 77–98

[30] Wang, Z., Lee, R.B.: New cache designs for thwarting software cache-based side channel attacks. In: Proceedings of the 34th annual international symposium on Computer architecture. ISCA '07, New York, NY, USA, ACM (2007) 494–505

[31] Manadhata, P.K., Wing, J.M.: A formal model for a system's attack surface. In: Moving Target Defense. (2011) 1–28

[32] Evans, D., Nguyen-Tuong, A., Knight, J.C.: Effectiveness of moving target defenses. In: Moving Target Defense. (2011) 29–48

[33] Popov, G., Mladenov, V.: Modeling diversity in recovery computer systems. In Mastorakis, N., Mladenov, V., Kontargyri, V.T., eds.: Proceedings of the European Computing Conference. Volume 27 of Lecture Notes in Electrical Engineering. Springer US (2009) 223–233

[34] Arlat, J., Kanoun, K., Laprie, J.C.: Dependability modeling and evaluation of software fault-tolerant systems. IEEE Trans. Comput. **39**(4) (April 1990) 504–513